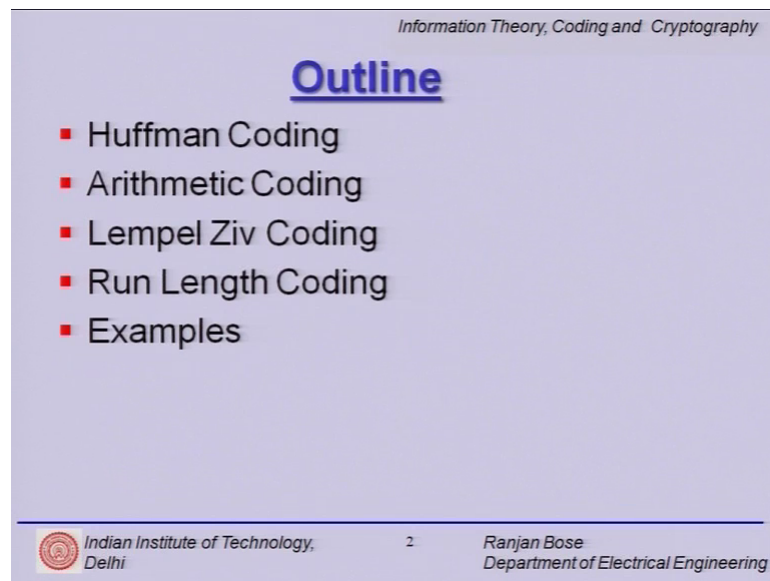


Information Theory, Coding and Cryptography
Dr. Ranjan Bose
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Module - 06
Source Coding
Lecture – 06

(Refer Slide Time: 00:23)



Information Theory, Coding and Cryptography

Outline

- Huffman Coding
- Arithmetic Coding
- Lempel Ziv Coding
- Run Length Coding
- Examples

Indian Institute of Technology, Delhi 2 Ranjan Bose
Department of Electrical Engineering

Hello and welcome to module 6 on source coding. Let us look at the outline for today's talk. We do a brief recap on Huffman Coding which we studied in the previous lecture, then we will look at something called Arithmetic Coding, followed by Lempel Ziv coding and then do something called Run Length Coding. So, these all are source coding algorithms and we would definitely look at some examples, so that is the agenda for the day

(Refer Slide Time: 00:53)

Information Theory, Coding and Cryptography

Recap

- Source Coding Theorem
- Efficiency of a code
- Huffman Coding
- Coding in blocks
- Examples

Indian Institute of Technology, Delhi 3 Ranjan Bose
Department of Electrical Engineering

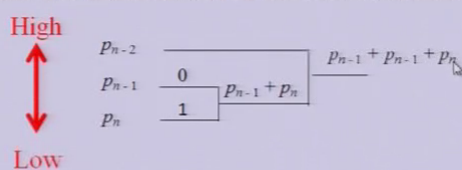
We start with a brief recap. We did look at source coding theorem, it tells us what is the best we can do in terms of compression. We then defined the efficiency of a code followed by some examples on Huffman Coding. We also saw that coding in blocks helps. So let us quickly refresh what we did regarding the Huffman Coding with an example.

(Refer Slide Time: 01:20)

Information Theory, Coding and Cryptography

Huffman Coding

- Each time we perform the combination of two symbols we reduce the total number of symbols by one.
- Whenever we tie together two probabilities (nodes) we label the two branches with a '1' and a '0'.
- Continue the procedure until only one probability is left (and it should be 1 if your addition is right!).
- This completes the construction of the Huffman tree.



The diagram illustrates a step in Huffman tree construction. On the left, three probabilities are listed vertically: p_{n-2} , p_{n-1} , and p_n . A red double-headed vertical arrow is positioned to the left of these probabilities, with the word "High" at the top and "Low" at the bottom. On the right, a tree structure is shown. A horizontal line connects p_{n-1} and p_n . Below this line, the label $p_{n-1} + p_n$ is placed. From the left end of this line, a vertical line goes down to the label "0". From the right end, a vertical line goes down to the label "1". These two lines then merge into a single line that goes up to the label $p_{n-1} + p_{n-1} + p_n$. A small arrow points to the right from the top of this final line.

Indian Institute of Technology, Delhi 4 Ranjan Bose
Department of Electrical Engineering

So, Huffman Coding requires you to first arrange the all the probabilities in a decreasing order. So, we write the symbols and write down the corresponding probabilities in

decreasing order, what we do is we combine the bottom 2 into a composite symbol with a probability $P_n + P_{n-1}$ and then we combine the next bottom two and so on so forth to add up the probabilities and this tree keeps elongating and the probabilities keep adding up till we reach 1.

(Refer Slide Time: 01:55)

Information Theory, Coding and Cryptography

Example

- Consider a DMS with seven possible symbols $x_i, i = 1, 2, \dots, 7$ and the corresponding probabilities
 $p_1 = 0.37, p_2 = 0.33, p_3 = 0.16, p_4 = 0.07, p_5 = 0.04, p_6 = 0.02, p_7 = 0.01$.
- We first arrange the probabilities in the decreasing order and then construct the Huffman tree.

Symbol	Probability
x_1	0.37
x_2	0.33
x_3	0.16
x_4	0.07
x_5	0.04
x_6	0.02
x_7	0.01

Indian Institute of Technology, Delhi 5 Ranjan Bose
Department of Electrical Engineering

(Refer Slide Time: 02:05)

Information Theory, Coding and Cryptography

Huffman Coding

Huffman Coding Tree

Indian Institute of Technology, Delhi 6 Ranjan Bose
Department of Electrical Engineering

We did this following example where we had the 7 symbols and the probabilities were arranged in decreasing order and we constructed this Huffman tree based on the probability. So, if you see the last two combined to give 0.03 and 0.04 and 0.03 add up to

0.07 and so and so forth and we grew up tree and we quickly see that it adds up to 1. And now finding out the code words is pretty simple going backwards, so for example for x 4, we start right at the model node and we follow the path back to the desired symbol. But each time we bifurcate, we add either a 1 or a 0. If you go down the ladder, we add up 1 ; if you go up the ladder, we add a 0 or we could interchange these.

So, for example, if explorer was to be encoded, I go down. So, 1 1 1 and up mean 0, so I will get x 4 to be 1 1 1 0 and this is called the Huffman Tree. You can always plug in the values of probabilities to determine H of X and R bar which is the average codeword length for this.

(Refer Slide Time: 03:21)

Information Theory, Coding and Cryptography


Example

$$H(X) = - \sum_{k=1}^7 P(x_k) \log_2 P(x_k) = 2.1152 \text{ bits}$$

$$\bar{R} = \sum_{k=1}^7 n_k P(x_k) = 1(0.37) + 2(0.33) + 3(0.16) + 4(0.07) + 5(0.04) + 6(0.02) + 6(0.01) = 2.1700 \text{ bits.}$$

The efficiency of this code is $\eta = (2.1152 / 2.1700) = 0.9747$

Symbol	Probability	Self Information	Codeword
x_1	0.37	1.4344	0
x_2	0.33	1.5995	10
x_3	0.16	2.6439	110
x_4	0.07	3.8365	1110
x_5	0.04	4.6439	11110
x_6	0.02	5.6439	111110
x_7	0.01	6.6439	111111


Indian Institute of Technology,
Delhi
7
Ranjan Bose
Department of Electrical Engineering

If you look at the formula H of X is nothing, but with a negative sign summation P x i log to the base 2 P x i over all is or in this case k and you get the values of H of X and similarly R bar is the average codeword lengths and you can get the value of the average codeword length. So, efficiency is the ratio is less than 1 in this case

So, this example is illustrative in the sense that the self information as shown in this table, they are not integers and we will see this is the biggest problem with prefix codes like Huffman which are unable to exactly map it.

(Refer Slide Time: 04:05)

Information Theory, Coding and Cryptography

Working blockwise


- In the above examples, encoding is done symbol by symbol. A more efficient procedure is to encode blocks of B symbols at a time. In this case the bounds of the source coding theorem becomes

$$BH(X) \leq \bar{R}_B < BH(X) + 1,$$

- since the entropy of a B -symbol block is simply $BH(X)$, and \bar{R}_B is the average number of bits per B -symbol block.
- We can rewrite the bound as

$$H(X) \leq \frac{\bar{R}_B}{B} < H(X) + \frac{1}{B}$$

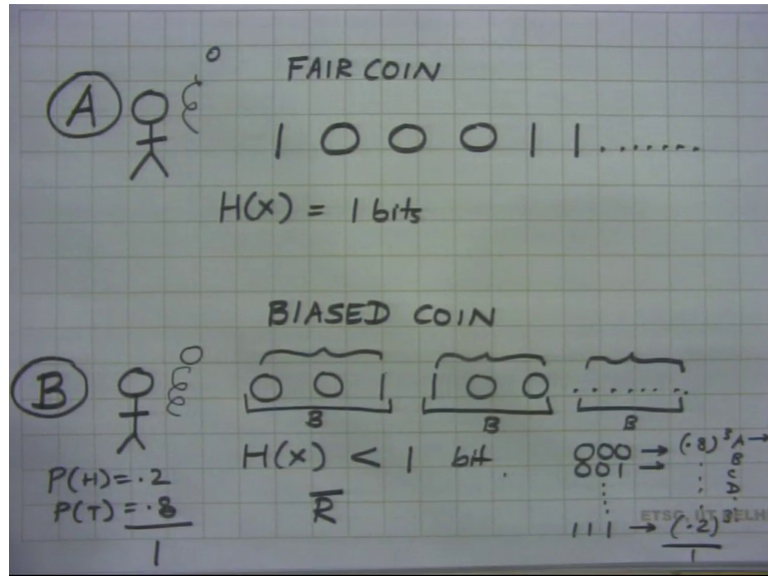
where $\frac{\bar{R}_B}{B} = \bar{R}$

 Indian Institute of Technology, Delhi 8 Ranjan Bose
Department of Electrical Engineering

We also saw in the last class that if we work block by block, then we have a better chance to do. So, and this is the reason why, suppose we encode a block of b symbols at a time right. Then we write the equivalent source coding theorem as B of $H(X)$ less than or equal to \bar{R}_B which is the average length for 1 block of b symbols and so on and so forth less than $BH(X) + 1$ coming back from the source coding theorem.

And we saw in the last class that if you divide both sides by B , you see that your $H(X)$ is the lower bound, but now the symbol per symbol if you find out the number of bits per symbol which is \bar{R}_B divided by B is limited upper bounded by $H(X) + \frac{1}{B}$. So, this is important as we increase the block size B , I can pinch it harder from the top and squeeze it to become closer and closer to $H(X)$ right that is the whole thing

(Refer Slide Time: 05:29)



If we look at a practical example of this, suppose we have our friend tossing a coin. So, we have a source A and this guy is tossing a coin, but this coin is a fair coin and every once in a while he tosses a coin; if a head comes it says 1, if a tail comes it says 0 0 0 1 1 it turns out this bit stream

But on the other hand I have another source B which is another person with a habit of chewing gum. So, he has stuck some chewing gum on 1 side of the coin and this time it is a biased coin, but his definition of entropy is the same. He tosses the coin once every second and spits out 1 0 0 1 randomly.

Now, if you see source A the H of X is 1 bit whereas, in this case H of X is less than 1. Consequently we should say that we should be able to use fewer number of bits on an average to send the output of source B as opposed to source A. But here I am at a dilemma because source B is also saying 1 bit every second because the person is tossing a coin and if a head or tail comes it speaks out 0 or 1.

So, the way to go about it is we can always look at say blocks of b and we already know that probability of head for example, is 0.2 and probability of tail is 0.2 0.8. For example, so they add up to 1, then it is easy to form a table right from 0 0 0 0 1 right up to 1 1 1 and we have this associated probabilities right. So, this will be 0.2 raised to the power 3 and this will be 0.8 raised to the power 3 and somewhere in the middle and they add up to 1.

So, you will see that we have now these are my new symbols and this source B does not have to say a 1 or a 0 or a 0 or a 1 depending upon the outcome of the head or a tail. It will be till 3 tosses and look at this combined symbol and it is a probability. So, these can be written as your ABCD so and so forth and they have the own Huffman Code

So, a variable length code will be actually used to send out these composite symbols and if you see on an average you will be able to have the average bits per symbols were to say will be less than 1 bit in this case. So, you actually require fewer than 1 bit to represent the outcome of this biased coin tossing, but this logic will not be true for fair coin where if you combine them in blocks of 3, 4, B does not matter.

They are all equiprobable and you will not be able to get any compression because of the combination into symbols. So, this is a simple example that illustrates why, if you do block by block coding you should be able to do things better.

(Refer Slide Time: 09:37)

Information Theory, Coding and Cryptography

Example

Symbol	Probability	Self Information	Codeword
x_1	0.40	1.3219	1
x_2	0.35	1.5146	00
x_3	0.25	2.0000	01

$H(X) = 1.5589$ bits
 $\bar{R} = 1.6000$ bits

$\eta = 0.9743$

Indian Institute of Technology,
Delhi
9
Ranjan Bose
Department of Electrical Engineering

We also get a simple example where we had three x_1 , x_2 , x_3 with associated probabilities and the self information and the code words and we calculated H of x to be 1.5589 bits and \bar{R} the average bits per symbol 1.6 bits and the efficiency was 0.97.

(Refer Slide Time: 10:01)

Information Theory, Coding and Cryptography

Example

Symbol Pairs	Probability	Self Information	Codeword
x_1x_1	0.1600	2.6439	10
x_1x_2	0.1400	2.8365	001
x_2x_1	0.1400	2.8365	010
x_2x_2	0.1225	3.0291	011
x_1x_3	0.1000	3.3219	111
x_3x_1	0.1000	3.3219	0000
x_2x_3	0.0875	3.5146	0001
x_3x_2	0.0875	3.5146	1100
x_3x_3	0.0625	4.0000	1101

$$2H(X) = 3.1177 \text{ bits} \Rightarrow H(X) = 1.5589 \text{ bits}$$

$$2\bar{R} = 3.1775 \text{ bits} \Rightarrow \bar{R} = 1.5888 \text{ bits}$$

$\eta = 0.9812$

➔

Better than

$\eta = 0.9743$

Indian Institute of Technology, Delhi

10

Ranjan Bose
Department of Electrical Engineering

Now, we did a simple experiment we made pairs. So, my B block length is 2. So, I have x_1x_1 , x_1x_2 and so on and so forth up to x_3x_3 and I can multiply the probabilities to get the net probabilities here. We calculate the self information and we do an a Huffman Coding to get this following table

Again if we calculate the average bits per symbol it comes out to be 1.588 and H of X is 1.5589 bits. Now, how does it compare with the previous one? Well previous one the efficiency was 0.97 and this time we have improved upon R efficiency

So, we are going closer and closer to your 1.55 in this case. We have been able to get to 1.58, but maybe 3 4 10 blocks at a time we can reach better and better of course, it comes at a cost. The cost is the computational complexity that is involved in both encoding and decoding


(Refer Slide Time: 11:13)

Information Theory, Coding and Cryptography

Problem with Prefix Codes

- Prefix codes try to **match** the self information of the symbols using codewords whose lengths are integers.
- The length matching may ascribe a codeword either **longer than** the self information or **shorter**.
- The exact match is possible if and only if the self information is in integral number of bits.

▶

 Indian Institute of Technology, Delhi 11 Ranjan Bose
Department of Electrical Engineering

So, that brings us to the problem of this prefix codes right. The Prefix code try to match the self information of the symbols with the length of the code words, but the lengths as we observed are integer. So, sometimes you are better sometimes you are worse off. The exact match is possible if and only if the self information is an integral number of bits

So, let us look at this more emphatically. So, when trying to see when does that happen; when does the exact number of bits and self information is an integer.


(Refer Slide Time: 11:54)

Information Theory, Coding and Cryptography

Problem with Prefix Codes

- The **exact match** is possible if and only if the self information is in integral number of bits.

Symbol	Probability	Self Information	Codeword	Codeword Length
x_1	2^{-1}	1	1	1
x_2	2^{-2}	2	00	2
x_3	2^{-3}	3	010	3
x_4	2^{-4}	4	0110	4
x_5	2^{-5}	5	01110	5
x_6	2^{-6}	6	011110	6
x_7	2^{-6}	6	011111	6

 Indian Institute of Technology, Delhi 12 Ranjan Bose
Department of Electrical Engineering


So, we look at this table quickly. Again we have this very special 7 symbols with very unique probabilities such that the self information is integer and the code word length is easy to match and they match exactly, but this is possible only because of a very unique probability distribution fine

(Refer Slide Time: 12:21)

Information Theory, Coding and Cryptography

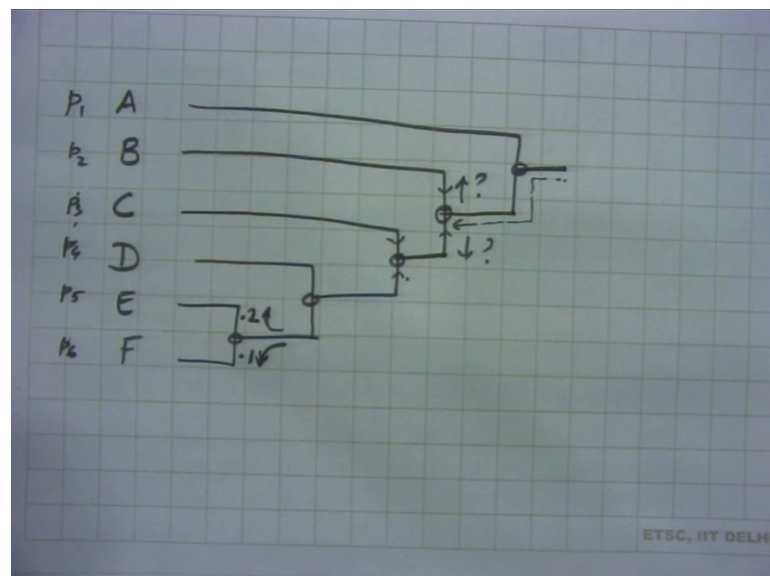
Problem with Prefix Codes

- If we consider the prefix codes being generated using a binary tree, the **decisions between tree branches** always take one bit.
- This is regardless of the fact whether the probabilities for the branches are 0.5/0.5 or 0.9/0.1.
- In the latter case it would theoretically take only 0.15 bits ($-\log_2(0.9)$) to select the first branch and 3.32 bits ($-\log_2(0.1)$) to select the second branch, making the average code length 0.467 bits ($0.9 \times 0.15 + 0.1 \times 3.32$).
- The Huffman code still needs **one bit for each decision**.

 Indian Institute of Technology, Delhi
13
Ranjan Bose
Department of Electrical Engineering

So, we now again look at why prefix code work, but they can work better what are the problems with prefix codes where do they fail. So, Prefix code as we saw are based on trees. So, Huffman Tree helps you to make the Huffman Code.

(Refer Slide Time: 12:49)



So, let us look at another example where you may have the symbols A B C D E and F and suppose they are decreasing order of probability. So, P 1, P 2 so and so forth right P 3, P 4, P 5, P 6 and you combine the bottom two to achieve this Huffman Tree ok. It is fairly mechanical.

But when whenever we go back and whenever we hit any of the bifurcations, we have to take a decision whether to go up or to go down right and we always attach 1 bit either which way? Whether we go up or go down, but what is telling us right because these two branches are the sum of the probabilities. So, suppose here the probabilities are 0.1 and 0.1, then going up and going down requires a decision of 1 bit. On the other hand, if it was 0.1 and 0.2, then things are not the same because information theoretically right; we should not ascribe the decision to go down or up does not warrant equal number of bits because they are not probabilistically equal

So, if you look at the slide, if we have a case of point 5 to 0.5 as opposed to 0.9 to 0.1. We should not give equal number of bits in both the cases, why? In the latter case of 0.9 and 0.1 because suppose these two branches finally, add up to 1 right. It would only take 0.15 bits to select the first branch and 3.32 bits to select the second branch because 0.1 is rarer than 0.9 probability and therefore, I need more number of bits to represent that the, but the Huffman code will always ascribe 1 bit per decision

(Refer Slide Time: 15:14)

Information Theory, Coding and Cryptography

D-adic Probability distribution

- Consider a DMS with seven possible symbols $x_i, i = 1, 2, \dots, 7$

Symbol	Probability	Self Information	Codeword	Codeword Length
x_1	2^{-1}	1	1	1
x_2	2^{-2}	2	00	2
x_3	2^{-3}	3	010	3
x_4	2^{-4}	4	0110	4
x_5	2^{-5}	5	01110	5
x_6	2^{-6}	6	011110	6
x_7	2^{-6}	6	011111	6

$H(X) = 1.9688$ bits
 $\bar{R} = 1.9688$ bits

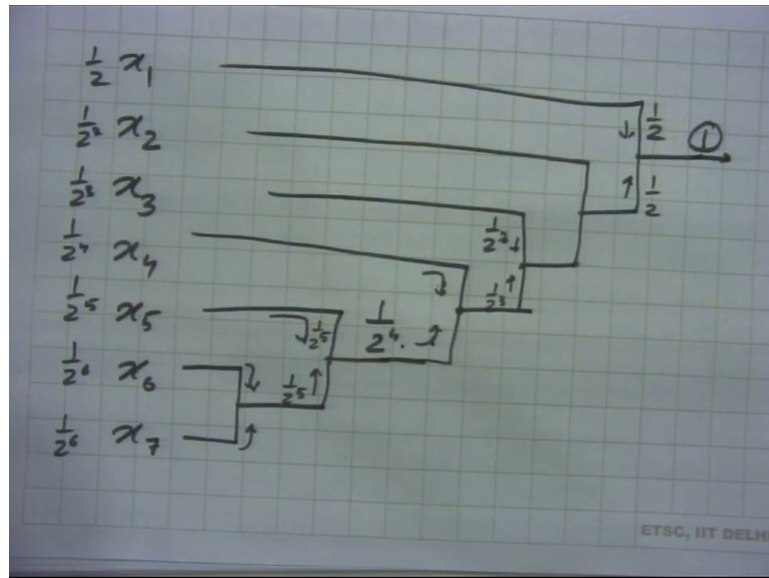
Indian Institute of Technology,
Delhi

14

Ranjan Bose
Department of Electrical Engineering

So, the D-adic probability distribution which we saw is the unique case where we have this decision at all stages you will have 1. So, if you look at this D-adic case, we have $x_1, x_2, x_3, x_4, x_5, x_6$ and x_7 and if you look at the probabilities it is $1/2$ raised to the power 6, $1/2$ raised to the power 5, $1/2$ raised to the power 4, cubed 2 squared and half very special distribution. Special in the sense both sides are equiprobable and they add up to $1/2$ raised to the power 5

(Refer Slide Time: 15:32)



Again when we go for combining both sides are equiprobable. So, you will get $1/2$ raised to the power 4. Again when we go for a combination very unique, but they are equiprobable. So, whenever we assign 1 bit going up or down, we are doing justice again and so on and so forth.


So, you will see at all stages, this is the only distribution that going up going down will add up to 1 and this tells us that why this D-adic distribution and only this one will allow us to really reach the efficiency of 1. This is an example, why we should be able to get the best possible compression in this case using Huffman Code. So, efficiency is clearly 1 in this D-adic probability distribution

(Refer Slide Time: 17:19)

Information Theory, Coding and Cryptography

Getting around the problem

- **Arithmetic coding** does not have this restriction.
- It works by representing the file to be encoded by an interval of real numbers between 0 and 1.
- Successive symbols in the message reduce this interval in accordance with the probability of that symbol.
- The more likely symbols reduce the range by less, and thus add fewer bits to the message.

 Indian Institute of Technology, Delhi 15 Ranjan Bose
Department of Electrical Engineering

So, we need to get around this problem and for that we propose this technique called arithmetic coding that does not have this restriction of this integer bits being assigned. We need to get on the real line because the probabilities can be as fine as possible ok. So, we definitely should not restrict ourselves to assigning code words right in the beginning. Let us play along the real line where we can have as much resolution as possible and only at the end should we talk about coding it into bits. Very simple logic, let us see how we do it.


So, it works by representing the file to be encoded. What do you mean by file to be encoded? Let us say, we have symbols your English language text. You can have images, you can have any other representation a string of symbols, but in this case let us just talk about a series of English alphabet. So, we have to compress this file which has several English alpha. So, what we do is we keep subdividing the line between 0 and 1 subsequently successively as a symbols coming and we go deeper and deeper into the line.

(Refer Slide Time: 18:50)

Information Theory, Coding and Cryptography

Arithmetic Coding

- Let our alphabet consists of only three symbols A , B and C
- Probabilities of occurrence $P(A) = 0.5$, $P(B) = 0.25$ and $P(C) = 0.25$.
- We first **divide** the interval $[0, 1)$ into three intervals proportional to their probabilities.
- Thus, the variable A corresponds to $[0, 0.5)$, the variable B corresponds to $[0.5, 0.75)$ and the variable C corresponds to $[0.75, 1.0)$.
- Note that the **lengths of these intervals** are proportional to their probabilities.

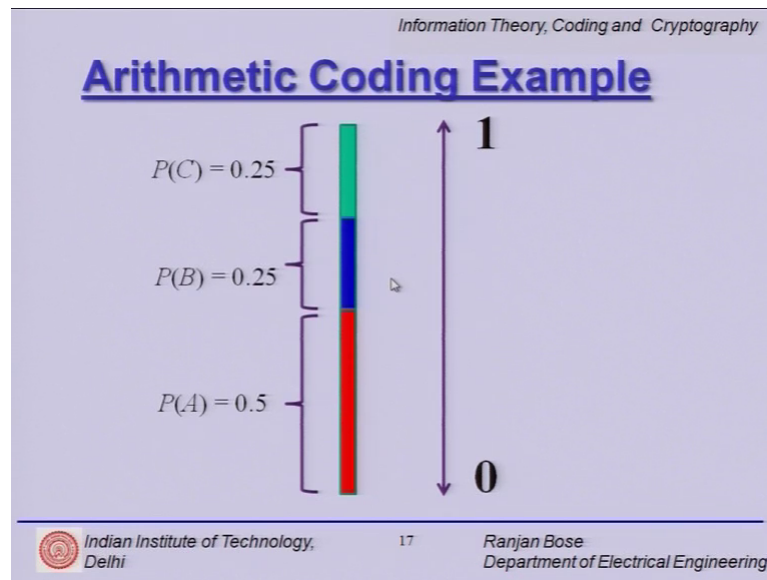
 Indian Institute of Technology, Delhi 16 Ranjan Bose
Department of Electrical Engineering

So, let us look at a simple example suppose we have only 3 symbols consisting of A , B and C . So, whatever be a bit stream, it is only as some combination of AA CC BA ABC or any other random combination. This is just a toy example for illustration.

Now, suppose we have the probabilities associated with it, only when the probabilities are different. Do we have a scope for compression? So, in this example let us say $P(A)$ is 0.5 , $P(B)$ is 0.25 and $P(C)$ is 0.25 again.

So, how do we begin? We have a real line from 0 to 1 , we divided into 3 intervals proportional to their probabilities. So, look numbers are resources. this real line is a resources it is a real estate. We would like to divide this real estate into this equal for $P(B)$ and $P(C)$ which have 0.25 probabilities whereas, $P(A)$ which has a higher probability will require 0.5 , so half the line.

(Refer Slide Time: 20:14)



So, let us graphically look at it. We have 0 and 1. This is a scratch pad, this is where we are going to work. Now we have this 3 probabilities 0.5, 0.25 and 0.25. So, why do not we divide it? So, half the line is the domain for P. What does it mean? From 0 to 0.5 is my region for A, 0.5 to 0.75 is my region for B and 0.75 to 1 is C. So, very mechanical I am just distributing the resource.

(Refer Slide Time: 21:08)

Information Theory, Coding and Cryptography

Arithmetic Coding Example

- Next, suppose the input symbol stream is
B A C A ...
- We first encode *B*. This is simply choosing the corresponding interval, i.e., [0.5, 0.75).
- Now, this interval is again subdivided into three intervals, proportional to the probabilities of occurrence.
- So, for the second step, the variable *A* corresponds to [0.5, 0.625), the variable *B* corresponds to [0.625, 0.6875) and the variable *C* corresponds to [0.6875, 1.0).

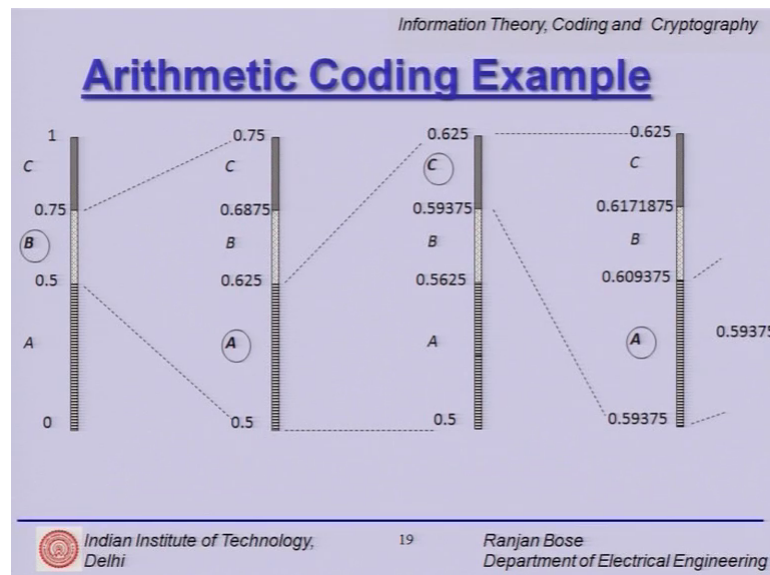
Indian Institute of Technology, Delhi 18 Ranjan Bose
Department of Electrical Engineering

Now, let us continue with an example. Suppose the bit stream to be encoded is B A C A so, B comes first. So, we focus our attention on B. Since B has come, if you consider the

previous example, B lies between 0.5 and 0.75. So, that is the thing we go, but then we scale at this 0.5 and 0.75 stretch it up and we will use this as a real estate to divide further between A B and C in the same ratios. So, B which was the first symbol that came in chose this subinterval 0.5 to 0.75.

Now, this sub interval will be further subdivided into 3 regions, A B and C half quarter and quarter depending upon the probabilities of A B and C. So, now 0.5 to 0.75, if you divide it half then it will become 0.5 to 0.625 belonging to A 0.625 to 0.6875 to B and 0.6875 to 0.75 for C.

(Refer Slide Time: 22:28)



So, let us look at it graphically again. We started with this real line 0 to 1, A had the line share of 0.5, B quarter of 8 and C of in quarter of the line. First symbol to come was B right. It is not very visible, but this B has come in first and so I take this section for B which belongs from 0.5 to 0.75 and stretch it.

So, the upper limit is 0.75 and the lower limit is 0.5 and again I quickly divided it into 3 portions equal to auto portion to the probability. So, A gets again half of it, but what is half of it 0.5 and 0.75, add it up and divide by 2, 0.625. So, that is the line for A and then subsequently divides 0.625 and 0.75. So, add them up and by 2 you get 0.6875 and so and so forth.

Now, please make an observation. But as we proceed, the boundaries are represented by numbers decimal points, but these are becoming larger and larger. So, earlier it was 0 0.5, 0.75 and 1. Now it is 0.5 0.6253 digits; 0.68754 digits, 0.752 digits. So, I am now requiring more and more numbers to represent my boundaries

Anyway so, I was working with B and then next number that comes in, the next symbol that comes in is A. So, the A comes in my boundaries become from 0.5 to 0.625, so I stretch it up. So, 0.5 remains 0.5; and this one becomes the upper limit. So, now, I am at this is my real estate to divide and rest is mechanical half of it. So, midway point nothing, but 0.5 plus 0.625 add them up and divide by 2. So, you get 0.5625 and then same with 0.59375 more surprise I am using more number of larger points after the decimal to represent the boundaries.

Next come see; so, I have already encoded B A and now go to C and it is the same mechanical I stretch it out to 0.625 as a top and 0.59375 as a bottom and when a comes in again I get the middle point and you get the feeling right. So, this is the way to go about doing this arithmetic coding and suppose, I say I have had enough why do not you not tell you what is the code word for B A C A right. What is the codeword? Well I am not giving you 1 symbol at a time, I am giving you the entire codeword for B A C A and the answer is 0.59375 any number between these two boundaries 0.59375 and 0.609375 any number pick at random is a valid codeword for B A C A

Student: (Refer Time: 26:09).

Pardon.

Student: So, there could be infinite numbers can be used.

Ok the question being asked is there can be infinite choices. The answer is no problem. At the end, I should be able to uniquely decoded and how do we decoded when it is the same thing? Suppose I say, look I have received 0.59375 at the receiver; tell me what was sent? That is it no problem.

Let us look at it when is 0.59375. So, I go my go to my real line between 0 and 1 and see where is it located where should this number 0.59375 lie? I said oh this number should lie in this region somewhere. I do not care where just this region. So, say oh this is the

region for B. So, first decoded symbol is B, but I am I d 1 with no. So, I go further. So, I divide this B section here and it still asked my question tell me where is 0.59375 would lie? And then I said oh 0.593, it is bigger than 0.5, but it is less than 0.6 to 5, it should lie in this region. I do not care where exactly it lies, it is in this region. Fair enough.

So, my second decoded symbol is A, but I am not done yet. So, I expanded further. Mechanical and I said where does 0.59375 lie? Now, I go ahead and say this is this region C right. It could have been anything above it also, but it is this way and then I said third symbol is C and I expand it further and I see where does it this lie and then I get the answer as A and you will be surprised to see that any one of the numbers in this region would give you the same answer, but if somebody does not tell me when to stop, I will keep going. After B A C A, I can give you a few more C C A B B D or D is not there, but A B B A.

So, I need a way to tell me when to stop. So, stopping flank or a q is required for arithmetic coding, but in this example we just found out that how arithmetic coding can be effectively done. So, my 0.59375 could be now represented as bits and sent. So, this.

Student: At the time up inputting, we have to know all the symbols at a time means all the symbols otherwise if you do not (Refer Time: 29:08) if I using arithmetic programming.

Ok. So, the question being asked is are we required to know the entire symbol stream to do that? Well the answer is yes because unless we keep going symbol by symbol and encoding, we will not be able to reach the final 0.59375 as an answer.

So, yes we decide to go by blocks and that helps us in the decoding process. So, it is a so, to say protocol for example, to say that I will only encode 50 symbols at a time. So, I take 50 symbols and then send anything I do not declared my codeword till I encode the 50 symbols and then I have some answer 0.593722192275 whatever and then that can be represented as bits and transmitted.

Student: Sir, the amount of number of bits you required to represent such a fine number, it can be more than the actually number needed to encode ABCD side?

So let me rephrase the question. The question being asked is, is it possible that the actual number bits required to represent this fine decimal number? We actually more than what we would have done otherwise say by doing Huffman Coding because after all I have these probabilities; So, nothing stops me from using a Huffman Code. The answer is no. Arithmetic coding offers a much better compression than your standard Huffman Coding and the reason is because it keeps working on this real line and it starts matching the occurrences of A B and C to as fine as possible.

This goes very fine, nowhere at any stage by putting a restriction on the self information being matched to the actual number of bits because nobody is talking about bits. This bits will come at the end. So, it will come out to be a winner as compared to Huffman Coding. You can try this out, but the reason why it works is it is not in a plagued by the problem of trying to match the interior number of bits in the codeword to the average self information that is the reason why this works much much better ok, but it is worthwhile trying out a simple example to compare the efficiency of arithmetic coding to Huffman Coding

So, both Arithmetic Coding and Huffman Coding form a part of the jpeg compression standards. But both of them have this requirement and the requirement is that you need the probabilities of A B and C A priori without that I cannot even take the first step


Now, it also does not have any. So, this say bias towards whether B A C are dependent on each other? We are treating them as independent; same with Huffman same with Arithmetic, but if it is an English language ing come in a pair, the are more frequent. So, q followed by u is more common it does not take care of that and the problem is that in reality who can tell me what is the probabilities of A B and C can be work around this problem.

(Refer Slide Time: 33:06)

Information Theory, Coding and Cryptography

The Lempel-Ziv Algorithm

- Huffman coding requires symbol probabilities.
- But most real life scenarios do not provide the symbol probabilities in advance (*i.e.*, the statistics of the source is unknown).
- Also, Huffman coding is optimal for a DMS source where the occurrence of one symbol does not alter the probabilities of the subsequent symbols.
- Huffman coding is **not** the best choice for a source with memory.
- We know that many letters occur in pairs or groups, like 'q-u', 't-h', 'i-n-g' etc.

 Indian Institute of Technology, Delhi 20 Ranjan Bose
Department of Electrical Engineering

And the answer was proposed by Lempel and Ziv in their algorithm. So, the shortcoming of Huffman is that it requires symbol probabilities. So, does arithmetic coding right

But in most real life scenarios symbol probabilities may not be known in advance or even if you do they may be changing over time the symbol may not be time invariant you would the probabilities might just change and Huffman Coding is great for discrete memoryless source ok. So, occurrence of one symbol does not affect the occurrence of the subsequent symbol and occurrence of a q will not influence whether a u will appear after that, but that is asking for too much


So, clearly Huffman Coding is not the choice not the best choice really for a source with memory where we have those sources in real life and we looked at this example q followed by u or t h or i n g do occur more frequently.

(Refer Slide Time: 34:27)

Information Theory, Coding and Cryptography

The Lempel-Ziv Algorithm

- The compression of an arbitrary sequence of bits is possible by coding a series of 0's and 1's as some *previous* such string (the prefix string) plus one new bit.
- Then, the new string formed by adding the new bit to the previously used prefix string becomes a potential prefix string for future strings.
- These variable length blocks are called **phrases**.
- The phrases are listed in a dictionary which stores the existing phrases and their locations.
- In encoding a new phrase, we specify the location of the existing phrase in the dictionary and append the new letter.

 Indian Institute of Technology, Delhi 21 Ranjan Bose
Department of Electrical Engineering

So, we are talking about compression of an arbitrary sequence of bits and it is possible by coding a series of 0 s and 1 as some previous such string plus 1 new bit. So, the basic philosophy of Lempel Ziv algorithm is as follows. It tries to represent any future string as something that is already arrived. So, it encodes future strings in terms of past strings


So, the new string found by adding a new bit to the previously used prefix string becomes a potential prefix string for future strings. So, thus these variable length blocks are called phrases and phrases are listed in a dictionary which stores the existing phrases and their location. So, all we have to say is location number so and so plus an innovation bit which makes a different. In encoding a new phrase, we specify the location of the existing phrase in the dictionary and append the new letter. We will look at an example.

(Refer Slide Time: 35:46)

Information Theory, Coding and Cryptography

Example

- Suppose we wish to code the string: **1010110110101011**.
- We will begin by **parsing** it into comma-separated phrases that represent strings that can be represented by a previous string as a prefix, plus a bit.
- The first bit, a 1, has no predecessors, so, it has a null prefix string and the one extra bit is itself:
1, 010110110101011
- The same goes for the 0 that follows since it can't be expressed in terms of the only existing prefix:
1, 0, 10110110101011

 *Indian Institute of Technology,
Delhi* 22 *Ranjan Bose
Department of Electrical Engineering*

Suppose we wish to encode this long string 1 0 1 0 1 1 0 1 1 and so and so forth. Ok this is our target bit stream to be encoded. We begin by doing something called parsing which is a comma separated phrase that represent strings that can be represented by a previous string


So, let us again revisit this string 1 0 1 0 1 and so forth and I put a comma after the first 1 right because we are starting from scratch right. So, it is a null prefix and a 1. And then we get a 0, so we express the second comma as after the 0. So, what it means is this 1, it is right in the beginning; we cannot really express it in terms of any of the previous bit. It is actually starting off with a 1 and a 0. In some realizations of Lempel Ziv, we already assumed 1 0 being the first two entries of your codebook, but here we are starting from scratch. Now, the real fun will begin.

(Refer Slide Time: 37:08)

Information Theory, Coding and Cryptography

Example

- So far our dictionary contains the strings '1' and '0'.
- Next we encounter a 1, but it already exists in our dictionary. Hence we proceed further. The following 10 is obviously a combination of the prefix 1 and a 0, so we now have:
1, 0, 10, 1101101010111
- Continuing in this way we eventually parse the whole string as follows:
1, 0, 10, 11, 01, 101, 010, 1011
- Now, since we found 8 phrases, we will use a three bit code to label the null phrase and the first seven phrases for a total of 8 numbered phrases (with the ninth and last phrase we found being expressed in the others, hence not needing to be numbered).

 Indian Institute of Technology, Delhi23Ranjan Bose
Department of Electrical Engineering

So, far a dictionary contains a 1 and a 0, it is not a surprise because we are binary encoding and everything will be represented in terms of 1's and a 0. But now let us look at the further parsing of the string. So, we already have 1 comma 0 comma and then we encountered a 1, but we know that 1 is already in my dictionary and that time location number 1, but 1 0 is not in my dictionary. So, this 0 makes this symbol, this is my new symbol 1 0 different from what is present. So, this is 0 which made it different is called the innovation bit.

So, now, I put a comma and then I proceed further. The next bit I encountered as a 1 I look into my dictionary. So, this is my small dictionary a horizontal dictionary which has 3 entries 1 0 and 1 0, I encountered a next 1. Is it present? Of course, it is.

So, I proceed and I look at this 1 1, Is it present? I scan my dictionary and say no 1 1 is not present. So, I put a comma behind 1 1 having put a comma, I now proceed further I encountered a 0. So, 0 is present of course, it is present that time location 2. I encountered a 1 so, 0 1. Is it present? No it is not. So, I put a comma because 0 1 is another parsed symbol and then I proceed further from this point onwards I encountered a 1. Is 1 present? Sure it is present at location number 1. Is 1 0 present? Oh yes it is present location number 3.

So, I proceed further 1 0 1. Is 1 0 1 present? No it is not. So, I put a comma after 1 0 1 and I proceed further along the same lines. So, I get 0 1 0 comma and then 1 0 1 1 comma and so and so forth

Now, if we make this observation as we go on parsing. These code words become longer and longer. At this point it is not fair to call them code words, but these parsed bits become longer and longer. So, 1 bit long, 2 bits long, 3 bit long, 4 bit long and so on so forth. So, we end up representing future bit streams in terms of the past bit streams.

(Refer Slide Time: 39:57)


Information Theory, Coding and Cryptography

Example

1, 0, 10, 11, 01, 101, 010, 1011

- Now, since we found 8 phrases, we will use a three bit code to label the null phrase and the first seven phrases for a total of 8 numbered phrases (with the ninth and last phrase we found being expressed in the others, hence not needing to be numbered).
- Next, we write the string in terms of the number of the prefix phrase plus the new bit needed to create the new phrase.
- We will use parentheses and commas to separate these at first, in order to aid our visualization of the process.
- The eight phrases can be described by:

(000,1),(000,0),(001,0),(001,1),(010,1),(011,1),(101,0),(110,1)

 Indian Institute of Technology, Delhi
24
Ranjan Bose
Department of Electrical Engineering

Now, if we complete this example, we can say that time is come to represent this past bit stream as fixed length code words. So, look at the first one. It is null because it really cannot be represented anything that comes before it. So, its 0 0 0 and innovation bit is 1, so 1. So, why do not we bring it closer ok, so that is better.

So, now this 1 is represented as 0 0 0 1. The next one will be represented by a codeword. Again 0 cannot be represented by anything before it. So, these are null 0 0 0 and the innovation bit is 0 which makes it different. So, 0 then we look at 1 0. So, 1 0 of course, has a 1 an innovation bit 0, 1 can be represented by a previous codeword, but where is it present? Location 1, 1 is represent at location 1. So, the first 3 bits are only indicating the location where which codeword, it says 0 0 1 location number 1 and what makes a different addition of a 0? So, 0 0 1 0.

Let us look at this 1 1. So, the first one is previously present, location number 1 represented by a location 0 0 1 and the innovation bit 1 just adds right here. Let us look at 0 1 0 1 is 0 followed by a 1 0 where is it located? Location 2, location 2 is represented by number 2 0 1 0 is number 2 and 1 makes it different. What about 1 0 1? 1 0 1 is represented with this is at location 3 1 0. So location 3, 0 1 1 and innovation bit is 1 and subsequently you can look at all of them.

Please note that 1 bit is also represented as 4 bits ; so, expansion, 1 bit is represented as 4 bits; expansion, 2 bits as 4 bits, 2 bits at 4 bits, 3 bit as 4 bits, 3 bit as 4 bits, 4 bit as 4 bits. So, far we are losing in terms of being efficient in terms of coding, but please note as we go along parsing longer and longer strings will be there, but all of them still be will be represented by 4 bits. So, we will start getting our compression advantage as we go along ok

So, the first few symbols will not actually yield compression, then end up expanding the number of bits see; the number of bits on top is larger than the number of bits which are originally present, but the case will drastically change as we go along.

(Refer Slide Time: 43:22)


Information Theory, Coding and Cryptography

Example

(000,1),(000,0),(001,0),(001,1),(010,1),(011,1),(101,0),(110,1)

- It can be read out as: (codeword at location 0,1), (codeword at location 0,0), (codeword at location 1,0), (codeword at location 1,1), (codeword at location 2,1), (codeword at location 3,1) ...
- Thus the coded version of the above string is:

00010000001000110101011110101101

 Indian Institute of Technology, Delhi
25
Ranjan Bose
Department of Electrical Engineering

So, what have we sent out as an encoded bit stream? Codeword at location 0 followed by 1, so 0 0 0 1 codeword at location 0 followed by 0 0 0 and so and so forth. So, if you just write it out without the commas, I send out the 00010000 and so, this is my encoded bit stream

So, binary string encoded in to another binary sting. What do I do at the decoder? If a decoder knows that we are doing a fixed length code. So, fixed length code means every 4 bits represents answer. So, I look at the first 4 bits, it is 0 0 0 1 within this. I also know that the first 3 bits represent the location and the next one is my innovation bit, but makes a different

So, location null location means it is a starting 1. So, I have the first entry 1 in my decoded. Does everybody see that? So, the first entry which I decode is the bit 1, first 4 bits are gone. I concentrate on my next 4 bits which is 0 0 0 0, I mean the first 3 bits represent the location null location and the 4 th is a innovation bit 0. So, clearly I write down immediately 0. So, the first two bits are 1 and 0 and the first 8 bits are decoded. So 9, 10, 11 and 12 bits, if you look at it and now you can actually see because these 2 are the same; the upper bit stream, lower bit stream, but that represented by commas. So, the decoder is actually looking at mentally, it is compartmentalizing it into 4 bits at a time.

So, we are now looking at the third bit, third to symbol it is 0 0 1 0. So, what is the decoder do? It moves at the first few bits are the location huh. The first location, we have found out is 0 0 1, but we already have decoded the first entry and what was that it was 1. So, it is 1 and the innovation bit is 0. So, it completely knows what was sent it is 1 0 and it again looks at the next 4 bits and its knows it is 1 1. But then it looks at this fifth one, it says 0 1 0 second location. What is in a second location bit 0 and what is the innovation bit 1? So, 0 1 so and so forth; so, decoding is pretty easy; It can be done very simply in Lempel Ziv of encoding algorithm.

Student: Sir.

Yes.

Student: At a code was longer, then it will be (Refer Time: 46:31) more bytes to represent the location. It would be able to (Refer Time: 46:37) location in 3 bits, then will be needing say 5 bit strange sorry its number. There also at the it would not be efficient.

So, let me tell the question, repeat the question. The question being asked is the number of bits used to represent the location depends on the length of the table, the size of the table right and as we form a longer and longer the encoding, we have size will increase.

Those only more and more codewords and eventually there will overflow. The observation is correct and so, we need to have a sufficiently long indicator for the location. But the second observation is not correct that whether it will be less efficient. In fact, it will still be a lot more efficient as we will see in the subsequent slides.

(Refer Slide Time: 47:31)

Information Theory, Coding and Cryptography

Example

(000,1),(000,0),(001,0),(001,1),(010,1),(011,1),(101,0),(110,1)

	Dictionary Location	Dictionary content	Fixed Length Codeword
1	001	1	0001
2	010	0	0000
3	011	10	0010
4	100	11	0011
5	101	01	0101
6	110	101	0111
7	111	010	1010
	-	1011	1101

Indian Institute of Technology, Delhi
 26
Ranjan Bose
Department of Electrical Engineering

So, if we are to complete this example, we have this dictionary location. Now we made this horizontal dictionary into a vertical 1. So, we have a dictionary location and we have a content and we will fixed length code. As observed, if the dictionary look length increases, the length the fixed length codeword will have to be longer because the location for the dictionary right will required more number of bits. So, 3 bits can only represent a dictionary of length 8, 4 bits can be represent 16, 5 bits 32, 6 bit 64. So, their growing exponentially just adding 1 bit can double the size of the dictionary

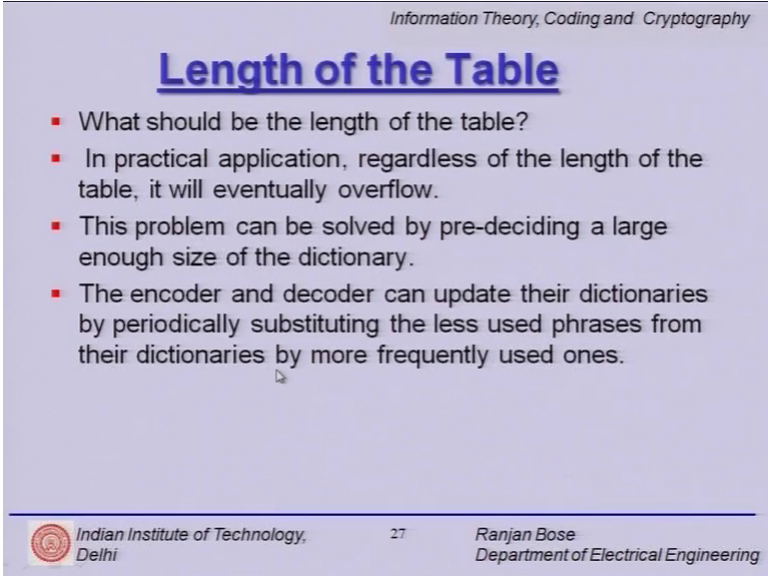
So, then itself, it tells us that this there is a lot of hope in this method, just adding 1 bit takes care. So, once, I mean 1 0 2 4 adding I one more bit increases the size of the dictionary by another thousand and then another two thousand and then 4 thousand.

So, I will be very happy when I am working with a long dictionary and in general we are working with larger dictionaries, but we have to fix this a priori that is the catch.

We cannot say that we will decide this fixed length codeword at a later stage. If it has to be 12, you tell me what is the anticipated size of the dictionary and fix the codeword

length accordingly. So, as you can see that this locations must correspond to the fixed over there. Here we have 7 and therefore, we need 3 bits to represent the fixed length codeword

(Refer Slide Time: 49:28)



Information Theory, Coding and Cryptography

Length of the Table

- What should be the length of the table?
- In practical application, regardless of the length of the table, it will eventually overflow.
- This problem can be solved by pre-deciding a large enough size of the dictionary.
- The encoder and decoder can update their dictionaries by periodically substituting the less used phrases from their dictionaries by more frequently used ones.

Indian Institute of Technology, Delhi 27 Ranjan Bose
Department of Electrical Engineering


So, let us just comment a little bit on the length of the table. What should the length of the table be because we just now saw that overflows are a reality. So, in practical application regardless of the length of the table, it will eventually overflow. This problem can be solved by pre-deciding a large enough size of the dictionary. Again it is the best bet we can have. The encoding decoder can update the dictionaries by periodically substituting the less used phrases from the dictionary by more frequently used 1's.

(Refer Slide Time: 50:04)

Information Theory, Coding and Cryptography

Run Length Encoding

- Run-length Encoding, or RLE is a technique used to reduce the size of a repeating string of characters.
- This repeating string is called a **run**.
- RLE can compress any type of data regardless of its information content, but the **content of data** to be compressed affects the compression ratio.
- Run-length encoding is supported by most bitmap file formats such as TIFF, JPG, BMP, PCX and fax machines.

 Indian Institute of Technology,
Delhi

28

Ranjan Bose
Department of Electrical Engineering

Now, we come to another encoding technique called the Run Length Encoding. It is an interesting source coding technique. It is a technique used to reduce the size of a repeating string of characters. So, it is not applicable everywhere, it is for only when we have runs of lengths runs of 0's or 1. So, what is the run? Well if you have a 0 0 0 0 0 many times over, it is a run of 0 similarly a run of 1.


So, run length coding could compress any type of data regardless of its information content right, but the content of the data the number of runs we have actually will decide the efficiency of run length coding and it is supported by most bitmap file formats like TIFF, JPG, BMP and so and so forth and very commonly used in fax machine because in fax machine where we have the page mostly blank. And character is written on top, if you represent a white pixel by a 0 and a black pixel by 1 with huge runs of 0's. So, it is very very efficient for fax machine.

(Refer Slide Time: 51:24)

Information Theory, Coding and Cryptography

Example

- Consider the following bit stream:
1111111111111110000000000000000001111.
- This can be represented as: fifteen 1's, nineteen 0's, four 1's, *i.e.*, (15, 1), (19, 0), (4, 1).
- Since the maximum number of repetitions is 19, which can be represented with 5 bits, we can encode the bit stream as (01111, 1), (10011, 0), (00100, 1).
- The compression ratio in this case is $18:38 = 1:2.11$.

 Indian Institute of Technology, Delhi 29 Ranjan Bose
Department of Electrical Engineering

Let us look at an example. Let us look at this strange bit stream. It has a long run of 1, then a run of 0, then a run of 1. We have taken this peculiar example because this effectively shows where run length encoding will work. So, what do we do we count the number of 1's and we say 15 1's count the number of 0's 1, 2, 3, 4 up to 19, 19 0's and then 4 1's. So, that this long bit stream is nothing, but 15 comma 1, 19 comma 0, 4 comma 1

So, it says how many and the after comma of what, how many of what how many of what. So, here, if this is the maximum run is of 19, then we can represent it with 5 bits. If you anticipate runs can be of the length thirty then also 5 bits or if the run lengths can be as large as 35, 40 up to 63 will require 6 bits to represent. But clearly the simple representation of this long bit stream shows the power of run length coding. In this case the compression is 1 is to 2.1. In this simple toy example, it is very high for fax machine applications.

(Refer Slide Time: 53:04)

Information Theory, Coding and Cryptography

Summary

- Revisited Huffman Coding
- Arithmetic Coding
- Lempel Ziv Coding
- Run Length Coding
- Examples

Indian Institute of Technology, Delhi 30 Ranjan Bose
Department of Electrical Engineering

So, let us now summarize today's lecture. We revisited the Huffman Coding and figured out why it works and where it fails. Then we looked at arithmetic coding which is a little bit more efficient and then we looked at Lempel Ziv coding which does not require a priori, the probabilities of the symbols and it will also works for sources with memory. And finally, we looked at an example of run length coding; that brings us to the end of module 6.